



GWU2U Driver (Windows VCP)

ユーザーガイド

UG1007-1.1J, 2021-12-23

著作権について(2021)

著作権に関する全ての権利は、**Guangdong Gowin Semiconductor Corporation** に留保されています。

GOWIN高云、Gowin、及びGOWINSEMIは、当社により、中国、米国特許商標庁、及びその他の国において登録されています。商標又はサービスマークとして特定されたその他全ての文字やロゴは、それぞれの権利者に帰属しています。何れの団体及び個人も、当社の書面による許可を得ず、本文書の内容の一部もしくは全部を、いかなる視聴覚的、電子的、機械的、複写、録音等の手段によりもしくは形式により、伝搬又は複製をしてはなりません。

免責事項

当社は、GOWINSEMI Terms and Conditions of Sale (GOWINSEMI取引条件)に規定されている内容を除き、(明示的か又は黙示的かに拘わらず)いかなる保証もせず、また、知的財産権や材料の使用によりあなたのハードウェア、ソフトウェア、データ、又は財産が被った損害についても責任を負いません。本文書における全ての情報は、予備的情報として取り扱われなければなりません。当社は、事前の通知なく、いつでも本文書の内容を変更することができます。本文書を参照する何れの団体及び個人も、最新の文書やエラッタ (不具合情報)については、当社に問い合わせる必要があります。

バージョン履歴

日付	バージョン	説明
2021/06/29	1.0J	初版。
2021/12/23	1.1J	3GWU2U 仮想シリアルポートのプログラミングガイドを更新。

目次

目次.....	i
図一覧	i
表一覧	ii
1 GWU2U の仮想シリアルポートドライバ	1
2 ドライバのアンインストール	3
3 GWU2U 仮想シリアルポートのプログラミングガイド	5
3.1 仮想シリアルデバイスを開く	5
3.2 タイムアウト設定	8
3.3 パラメータの設定	9
3.4 シリアルポートバッファのクリア	11
3.5 シリアルポートエラーのクリア	11
3.6 仮想シリアルデバイスを介したデータ送信(同期)	11
3.7 仮想シリアルデバイスを介したデータ受信(同期)	12
3.8 シリアルポートの非同期読み出しおよび書き込み	13
用語、略語	24
テクニカル・サポートとフィードバック	25

図一覧

図 1-1 List All Devices	1
図 1-2 デバイスを選択	2
図 1-3 ドライバーを選択.....	2
図 2-1 デバイスマネージャーを開く	3
図 2-2 ドライバーのアンインストール.....	4

表一覽

表 A-1 用語、略語	24
-------------------	----

1 GWU2U の仮想シリアルポート ドライバ

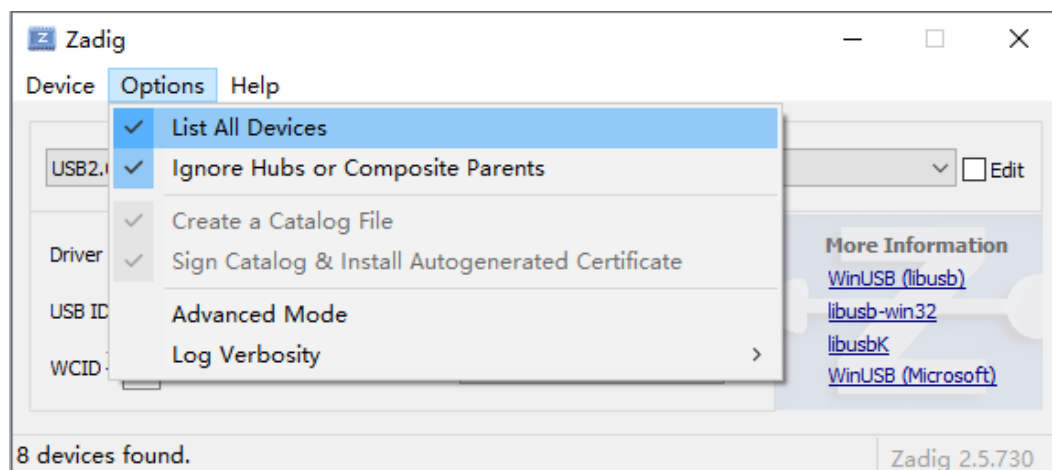
GWU2U は、USB 仮想シリアルポートドライバ(usbser.sys)を使用してプログラムすることにより、データの送受信や、ボーレート、パリティビット、ストップビット、ビット幅などのパラメータの設定を実現することができます。

ドライバをインストールするには、Gowin が提供するドライバ・インストール・ツールまたはオープンソースのドライバ・インストーラツール [Zadig](https://zadig.akeo.ie/)(<https://zadig.akeo.ie/>)を使用できます。ドライバをインストールするには、管理者権限が必要です。

Zadig を使用してドライバをインストール

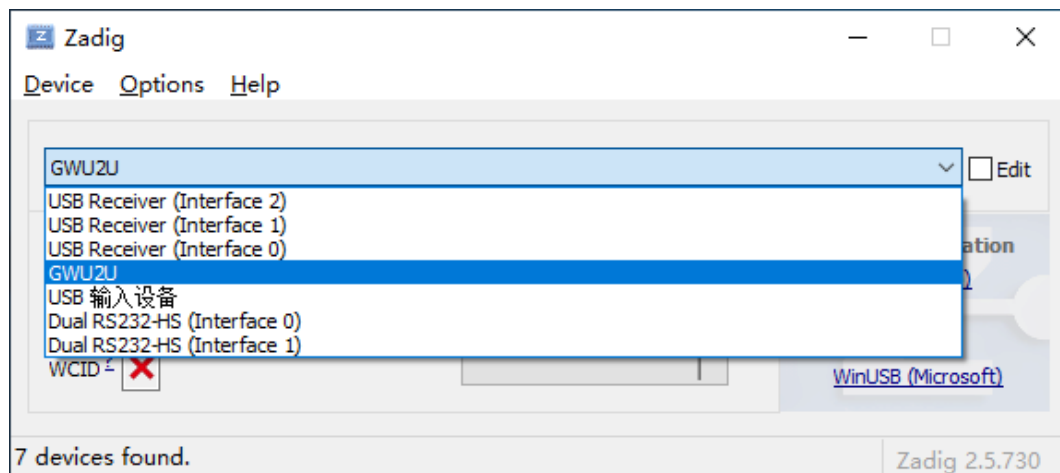
GWU2U デバイスをコンピュータの USB インターフェースに接続し、Zadig をダブルクリックして開き(管理者権限が必要)、Options > List All Device をチェックすると、コンピュータに接続されているすべての USB デバイスが一覧表示されます。

図 1-1 List All Devices



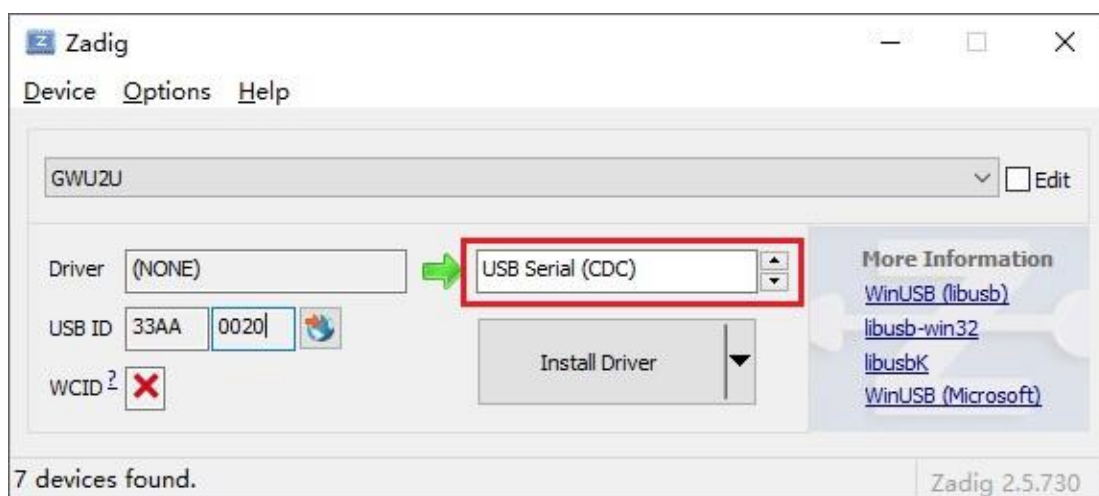
デバイス GWU2U を選択して、ドライバーをインストールします。

図 1-2 デバイスを選択



インストールするドライバーを選択します。仮想シリアルポート(VCP)を使用する場合、USB Serial (CDC)を選択してください。

図 1-3 ドライバーを選択



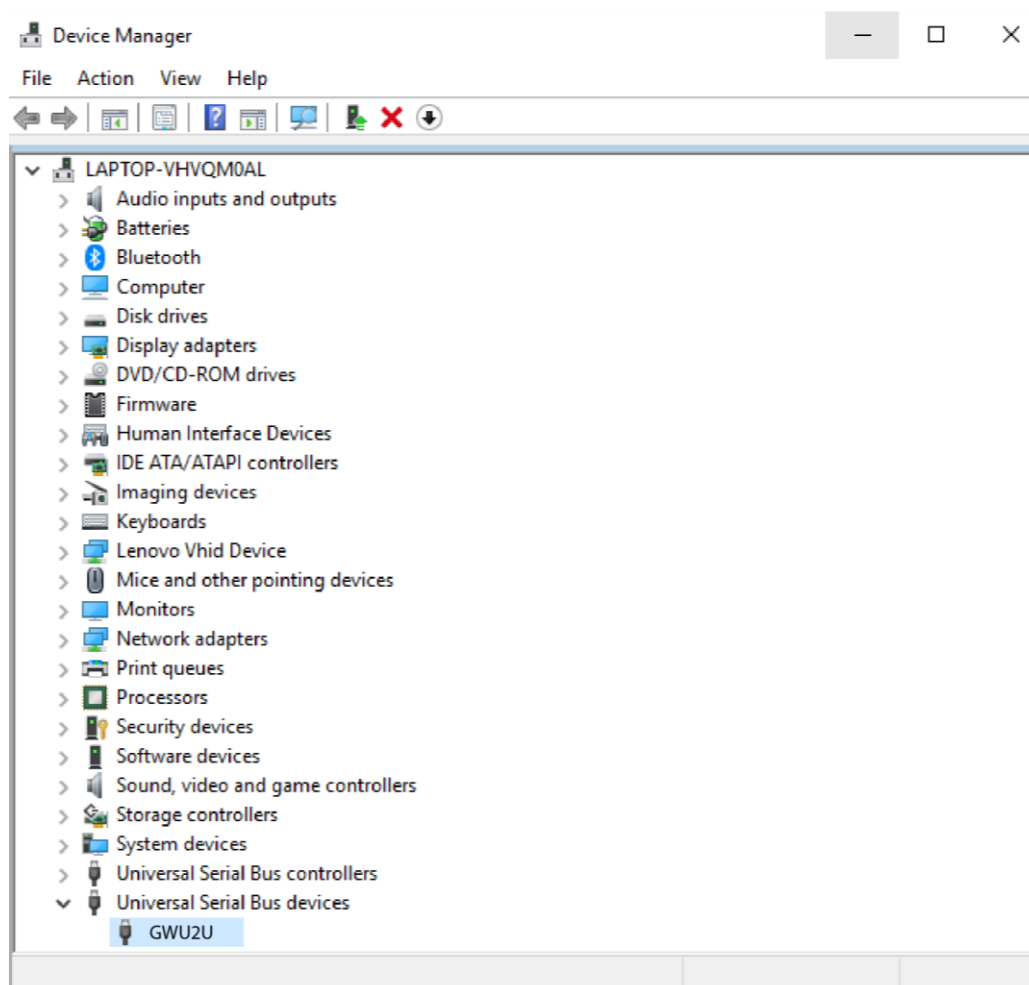
“Install Driver” ボタンをクリックして、ドライバーをインストールします。

なお、ドライバーが現在インストールされていない場合は“Install Driver”、他のドライバーがインストールされている場合は“Replace Driver”と表示されます。

2 ドライバーのアンインストール

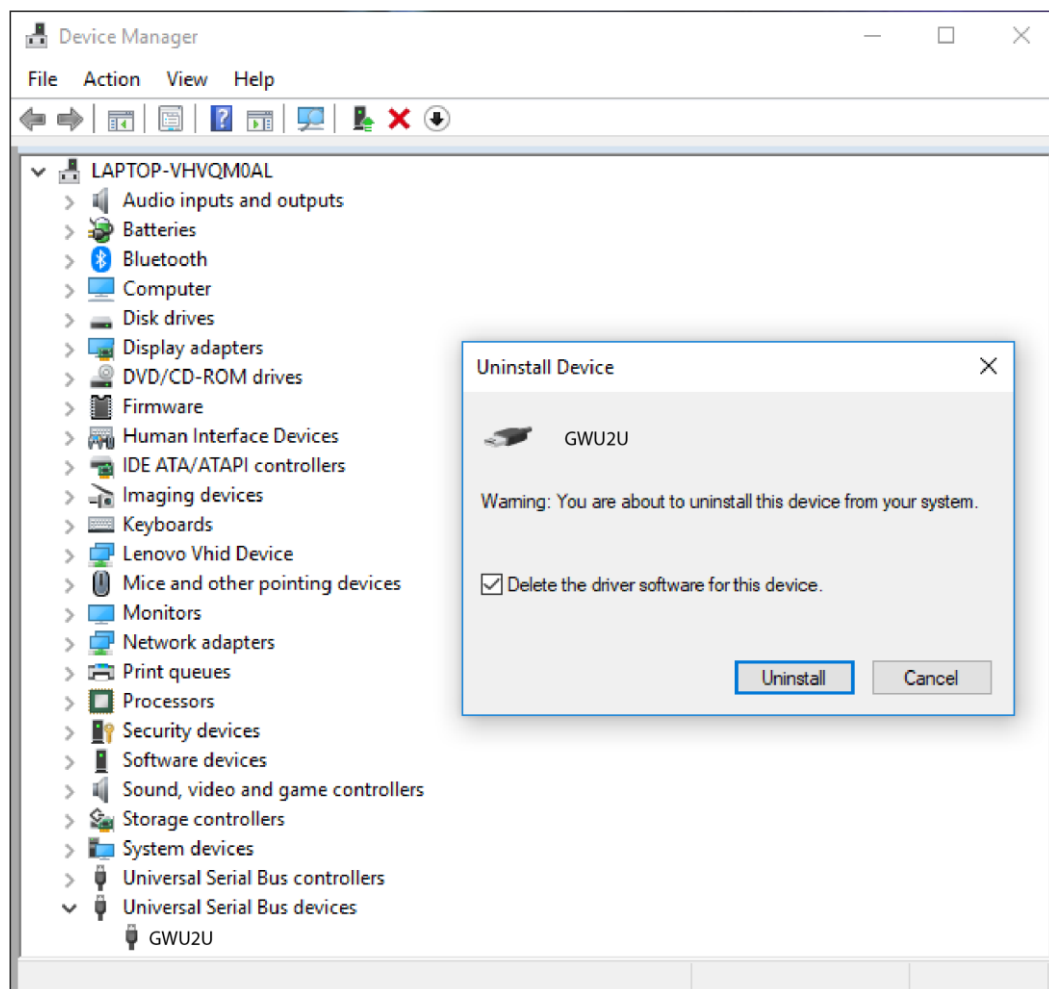
ドライバーをアンインストールするときは、GWU2U デバイスをコンピュータに接続し、Windows デバイスマネージャーを開いて、“ユニバーサル シリアル バス デバイス”リストで **Gowin USB Serial** デバイスを特定します。デバイス名を右クリックして、右クリックメニューの“削除”オプションを選択します。

図 2-1 デバイスマネージャーを開く



ポップアップダイアログボックスで、“Delete the driver software for this device(このデバイスのドライバー ソフトウェアを削除する)” にチェックを入れてアンインストールします。

図 2-2 ドライバーのアンインストール



3 GWU2U 仮想シリアルポートの プログラミングガイド

Windows のライブラリ関数を使用して、GWU2U の仮想シリアルデバイスを操作します。プログラムするときは、ヘッダーファイル<windows.h>をインクルードします。

3.1 仮想シリアルデバイスを開く

Windows ライブラリ関数 **CreateFile()** を使用して、仮想シリアルデバイスを開き、シリアルポートへのハンドルを返します。関数インターフェースは次のように定義されています。

```
HANDLE WINAPI CreateFile(  
    _In_ LPCTSTR lpFileName, //開くまたは作成するファイルの名前  
    _In_ DWORD dwDesiredAccess, //アクセス方法  
    _In_ DWORD dwShareMode, //共有モード  
    _In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes, //  
    セキュリティ属性  
    _In_ DWORD dwCreationDisposition, //開くファイルがすでに存在  
    する、または存在しない場合のアクションを指定します  
    _In_ DWORD dwFlagsAndAttributes, //ファイル属性とフラグ  
    _In_opt_ HANDLE hTemplateFile //テンプレートファイルへのハンド  
ル  
);
```

関数パラメータは次のように定義されます。

lpFileName : 開くまたは作成するファイルの名前。

dwDesiredAccess : アクセス方法 :

- 0 はデバイスクエリアクセス
- **GENERIC_READ** は読み出しアクセス
- **GENERIC_WRITE** は書き込みアクセス

dwShareMode : 共有モード :

- 0 は、ファイルを共有できず、ファイルを開くための他の操作が失敗することを示します。
- **FILE_SHARE_READ** は、他の読み出し操作が許可されることを示します。
- **FILE_SHARE_WRITE** は、他の書き込み操作が許可されることを示します。
- **FILE_SHARE_DELETE** は、他の削除操作が許可されることを示します。

lpSecurityAttributes : セキュリティ属性。 **SECURITY_ATTRIBUTES** 構造体へのポインタ。

dwCreationDisposition : ファイルを作成または開くときのアクション。

- **OPEN_ALWAYS** : ファイルを開き、ファイルが存在しない場合は作成します。
- **TRUNCATE_EXISTING** : ファイルを開き、ファイルをクリアします (したがって、**GENERIC_WRITE** 権限が必要です)。ファイルが存在しない場合は失敗します。
- **OPEN_EXISTING** : ファイルを開きます。ファイルが存在しない場合は失敗します。
- **CREATE_ALWAYS** : ファイルを作成します。ファイルがすでに存在する場合はクリアされます。
- **CREATE_NEW** : ファイルを作成します。ファイルが存在する場合は失敗します。

dwFlagsAndAttributes : ファイルのフラグと属性 :

- **FILE_ATTRIBUTE_NORMAL** : 一般属性。
- **FILE_FLAG_OVERLAPPED** : 非同期 IO フラグ、このフラグが指定されていない場合、デフォルトは同期 IO です。
- **FILE_ATTRIBUTE_READONLY** : ファイルは読み出し専用です。
- **FILE_ATTRIBUTE_HIDDEN** : ファイルは非表示です。
- **FILE_FLAG_DELETE_ON_CLOSE** : ファイルはファイルハンドルが閉じられた後に削除されます。

- その他のフラグと属性については、Microsoft の公式ドキュメントを参照してください。

hTemplateFile : ファイルのハンドル。ファイルは **GENERIC_READ** アクセスモードで開く必要があります。このパラメータが **NULL** でない場合、**hTemplateFile** に関連付けられたファイルの属性とフラグがファイルの作成に使用されます。既存のファイルを開く場合、このパラメータは無視されます。

CreateFile()を使用してシリアルポートを開く場合の注意事項 :

- **lpFileName** の形式 : **COM10** 未満の場合、シリアルポート番号(例えば、**COM1**)を直接入力します。**COM10** 以上の場合、シリアルポート名の形式は~~\\\\.\\~~**COM10** である必要があります。
- **dwShareMode** 共有モードは **0** である必要があります。つまり、シリアルポートは排他モードである必要があります。
- **dwCreationDisposition** が開かれるときのアクションは **OPEN_EXISTING** である必要があります。つまり、シリアルポートが存在する必要があります。

シリアルポートを閉じるときは、**CloseHandle()**関数を呼び出します。この関数のパラメータはシリアルポートハンドルです。

例 :

```
HANDLE hCom_1 =
CreateFile (
    ComName,
    GENERIC_READ | GENERIC_WRITE,
    0,
    NULL,
    OPEN_EXISTING,
    FILE_FLAG_OVERLAPPED, //OVERLAPPED モードでシリアルポートを
                           開いて非同期転送します。同期送信を使用する場合は、パラメータ 0 を使用
                           します
    NULL
); // シリアルデバイスを開きます

if (hCom_1 == INVALID_HANDLE_VALUE)
{
```

```
int a = GetLastError();

printf("Error: %d¥n", a);

return -1;

}

CloseHandle(hCom_1); // シリアルデバイスを閉じます
```

3.2 タイムアウト設定

ReadFile() と **WriteFile()** を呼び出してシリアルポートの読み出しと書き込みを行う場合、非同期操作が指定されていない場合、読み出しと書き込みは常に指定されたサイズのデータを待機します。この場合、読み出しと書き込みのタイムアウト期間を設定する必要があります。

SetCommTimeouts() を呼び出してシリアルポートの読み出しおよび書き込みタイムアウト時間を設定します。**GetCommTimeouts()** は現在のタイムアウト設定を取得できます。通常、最初に **GetCommTimeouts** を使用して現在のタイムアウト情報を **COMMTIMEOUTS** 構造体を取得し、この構造体をカスタマイズしてから、**SetCommTimeouts()** を呼び出して設定します。

COMMTIMEOUTS のメンバ変数は次のとおりです。

ReadIntervalTimeout : 2 文字間の最大遅延。シリアルポートデータを読み出すときに、2 文字の転送間の時間差がこの時間を超えると、読み出し関数は既存のデータを返します。0 に設定すると、このパラメータは機能しません。通信回線に 2 文字が到着する時間の間の最大遅延をミリ秒単位で指定します。**ReadFile** 操作中、この期間は最初の文字の受信から始まります。受信した 2 文字の時間間隔がこの値を超えると、**ReadFile** 操作が完了し、バッファリングされたすべてのデータが返されます。**ReadIntervalTimeout** が 0 の場合、この値は機能しません。値が **MAXDWORD** で、**ReadTotalTimeoutConstant** と **ReadTotalTimeoutMultiplier** の両方の値が 0 の場合、指定された読み出し操作は受信した文字を、文字が受信されていないくてもすぐに返します。

ReadTotalTimeoutMultiplier : 各文字読み出し間のタイムアウト。累積値をミリ秒単位で指定します。これは、読み出し操作中のタイムアウトの総数を計算するために使用されます。読み出し操作ごとに、この値に読み出されるバイト数が乗算されます。

ReadTotalTimeoutConstant : シリアルポートデータを一回読み出すための固定タイムアウト。したがって、一回のシリアルポート読み出し操作では、タイムアウトは “**ReadTotalTimeoutMultiplier** x 読み出されたバイト数 + **ReadTotalTimeoutConstant**” になります。**ReadIntervalTimeout** を

MAXDWORD に設定し、ReadTotalTimeoutMultiplier と ReadTotalTimeoutConstant を 0 に設定することは、読み出し操作が入力バッファに格納されている文字をすぐに返すことを意味します。

WriteTotalTimeoutMultiplier : 各文字書き込み間のタイムアウト。

WriteTotalTimeoutConstant: シリアルポートデータを一回書き込むための固定タイムアウト。したがって、一回のシリアル書き込み操作では、タイムアウトは” WriteTotalTimeoutMultiplier x 書き込まれたバイト数 + WriteTotalTimeoutConstant” になります。

例 :

```
//Set read timeout

COMMTIMEOUTS timeouts;

GetCommTimeouts(hCom_1, &timeouts);

timeouts.ReadIntervalTimeout = 0;

timeouts.ReadTotalTimeoutMultiplier = 0;

timeouts.ReadTotalTimeoutConstant = 6000;

timeouts.WriteTotalTimeoutMultiplier = 0;

timeouts.WriteTotalTimeoutConstant = 0;

SetCommTimeouts(hCom_1, &timeouts);


//Set read/write buffer size

static const int nBufSize = 32768;

if(!SetupComm(hCom_1, nBufSize, nBufSize))

{

    printf("SetupComm() failed, comm port closed...¥n");

    CloseHandle(hCom_1);

    return -1;

}
```

3.3 パラメータの設定

Windows には、シリアルポートパラメータを設定するための特別なデータ構造 DCB と特別なインターフェース関数 GetCommState()/SetCommState()があります。

DCB は、シリアル通信デバイスの設定に使用されるデータ構造です。シ

リアルポートの基本設定の変数は次のとおりです。

BaudRate : ボーレート設定。設定できる最大値は **256000** です。

Parity : パリティ設定。NOPARITY、ODDPARITY、EVENPARITY、MARKPARITY、SPACEPARITY に設定できます。

ByteSize : 1 バイトを送信するためのビット数。

StopBits : ストップビット設定。1 ストップビット(ONESTOPBIT)、1.5 ストップビット(ONE5STOPBITS)、2 ストップビット(TWOSTOPBITS) に設定できます。

例 :

```
//Set comm port configurations

DCB dcb;

if (!GetCommState(hCom_1, &dcb))
{
    printf("GetCommState() failed, comm port closed...¥n");
    CloseHandle(hCom_1);
    return -1;
}

//Configuration settings
int nBaud = 115200;
dcb.DCBlength = sizeof(DCB);
dcb.BaudRate = nBaud;          //ボーレート
dcb.Parity    = NOPARITY;     //パリティ
dcb.ByteSize  = 8;            //ビット幅
dcb.StopBits  = TWOSTOPBITS;  //ストップビット

if (!SetCommState(hCom_1, &dcb))
{
    printf("SetCommState() failed¥n");
    CloseHandle(hCom_1);
    return -1;
}
```

3.4 シリアルポートバッファのクリア

Windows には、読み出しおよび書き込み操作を停止し、読み出しおよび書き込みバッファをクリアするための **PurgeComm()**関数があります。シリアルデータを初めて読み出し書き込みする前、またはシリアルポートを長期間使用しなかった場合、またはシリアルポートにエラーがある場合は、最初に読み出しおよび書き込みバッファをクリアする必要があります。例：

```
DWORD dwError;

COMSTAT cs;

if(!ClearCommError(hCom_1, &dwError, &cs))
{
    printf("ClearCommError() failed\n");

    CloseHandle(hCom_1);

    return -1;
}
```

3.5 シリアルポートエラーのクリア

Windows には、通信のエラーをクリアして現在の通信ステータスを取得するための **ClearCommError()**関数があります。読み出しと書き込みを行う前に、**ClearCommError()**を呼び出してエラーをクリアし、バッファ内のデータサイズを取得できます。例：

```
DWORD dwError;

COMSTAT cs;

if(!ClearCommError(hCom_1, &dwError, &cs))
{
    printf("ClearCommError() failed\n");

    CloseHandle(hCom_1);

    return -1;
}
```

3.6 仮想シリアルデバイスを介したデータ送信(同期)

Windows が提供する関数 **WriteFile()**を使用してデータを送信できます。実行が成功した場合は **TRUE** を返し、失敗した場合は **FALSE** を返します。

データを同期的に送信する場合は、関数パラメータの **OVERLAPPED** デ

ータ構造のアドレスを **NULL** に設定する必要があります。

例：

```
BOOL bErrorFlag = FALSE;

char DataBuffer[] = "This is some test data to write to the virtual
com port.¥r¥n";

DWORD dwBytesToWrite = (DWORD)strlen(DataBuffer);

DWORD dwBytesWritten = 0;

bErrorFlag =
WriteFile (
    hCom_1,    // シリアルポートハンドル
    DataBuffer,    // 送信するデータを保存するアドレス
    dwBytesToWrite, // 送信するデータバイト数
    &dwBytesWritten, // 実際の送信データ長さを保存する変数のアドレス
    NULL        // OVERLAPPED データ構造アドレス、同期送信の場
                // 合は NULL
);
```

3.7 仮想シリアルデバイスを介したデータ受信(同期)

Windows が提供する関数 **ReadFile()**を使用してデータを受信できます。実行が成功した場合は **TRUE** を返し、失敗した場合は **FALSE** を返します。

データを同期的に受信する場合は、関数パラメータの **OVERLAPPED** データ構造のアドレスを **NULL** に設定する必要があります。

例：

```
char buf[101];

BOOL bErrorFlag = FALSE;

DWORD nLenOut = 0;

bErrorFlag =
ReadFile (
    hCom_1,    // シリアルポートハンドル
    buf,        // 受信するデータを保存するアドレス
    100,        // 受信するデータバイト数
    &nLenOut,    // 実際の受信データ長さを保存する変数のアドレス
```

```
        NULL          // OVERLAPPED データ構造アドレス、同期受信の場合は NULL
    );

    if (bErrorFlag)
    {
        if (nLenOut)
        { // 成功
        }
        else
        { // タイムアウト
        }
    }
    else
    { // 失敗
    }
}
```

3.8 シリアルポートの非同期読み出しおよび書き込み

シリアルポートの非同期読み出しと書き込みの方法は、同期読み出しと書き込みの方法と似ていますが、シリアルポートをオーバーラップ方法で開く必要がある点が異なります。オーバーラップ操作をすぐに完了できない場合、`WaitCommEvent()`は **FALSE** を返し、`GetLastError()`は **ERROR_IO_PENDING** を返します。これは、操作がバックグラウンドで実行されていることを示します。`WaitCommEvent` が返す前に、パラメータオーバーラップ構造の **hEvent** メンバーは **no signal** 状態に設定されます。イベントが発生したとき、またはエラーが発生したときに **signal** 状態に設定された場合、アプリケーションは待機関数(`WaitForSingleObject`、`WaitForSingleObjectEx` など)を呼び出してイベントオブジェクトの状態を判別でき、`WaitCommEvent()`のパラメータ **lpEvtMask** は発生した特定のイベントを保存します。

オーバーラップ操作が完了したかどうかを待機または判断するには、2つの方法があります：

- 1 つは、`WaitForSingleObject()`を使用して、読み出しおよび書き込み関数の **OVERLAPPED** タイプパラメータの **hEvent** メンバーを待機することです。`ReadFile` 関数と `WriteFile` 関数が呼び出されると、メンバー

は自動的に **no signal** 状態に設定されます。オーバーラップ操作が完了すると、このメンバー変数は自動的に **signal** 状態に設定されます。

- もう 1 つの方法は、**GetOverlappedResult()** を呼び出してオーバーラップ操作の状態を取得し、オーバーラップ操作が完了したかどうかを判断することです。

以下は、完全な仮想シリアルポートの非同期読み出しおよび書き込みの例です。

```
//ヘッダーファイル
```

```
#include <stdio.h>
```

```
#include <windows.h>
```

```
DWORD WINAPI ThreadRxMsg(LPVOID lpParameter);
```

```
void getComName(char* ComNamePtr, unsigned int ComNum);
```

```
void printBuf(char *buf, unsigned int bufSize);
```

```
char ComName[13];
```

```
HANDLE hCom_1;
```

```
int main(int argc, char* argv[])
```

```
{
```

```
    DWORD dwError;
```

```
    unsigned int ComNum = 26;
```

```
    getComName(ComName, ComNum);
```

// 仮想シリアルポートデバイスを OVERLAPPED モードで開きます。シリアルポート名は COM26 です。

```
    hCom_1 =
```

```
    CreateFile (
```

```
        ComName,
```

```
        GENERIC_READ | GENERIC_WRITE,
```

```
        0,
```

```
    NULL,  
    OPEN_EXISTING,  
    FILE_FLAG_OVERLAPPED,  
    0  
);  
  
if(hCom_1 == INVALID_HANDLE_VALUE)  
{  
    int a = GetLastError();  
    printf("Error : %d¥n", a);  
    return -1;  
}  
  
// タイムアウト設定  
COMMTIMEOUTS timeouts;  
GetCommTimeouts(hCom_1,&timeouts);  
timeouts.ReadIntervalTimeout = 0;  
timeouts.ReadTotalTimeoutMultiplier = 0;  
timeouts.ReadTotalTimeoutConstant = 6000;  
timeouts.WriteTotalTimeoutMultiplier = 0;  
timeouts.WriteTotalTimeoutConstant = 0;  
SetCommTimeouts(hCom_1,&timeouts);  
  
// 読み出し書き込みバッファのサイズの設定  
static const int nBufSize = 32768;  
if(!SetupComm(hCom_1,nBufSize,nBufSize))  
{  
    printf("SetupComm() failed, comm port closed...¥n");  
    CloseHandle(hCom_1);  
    return -1;  
}
```

```
    }

    // シリアルポートのパラメータの設定
    DCB dcb;
    if (!GetCommState(hCom_1, &dcb))
    {
        printf("GetCommState() failed, comm port closed...%n");
        CloseHandle(hCom_1);
        return -1;
    }

    int nBaud = 115200;
    dcb.DCBlength = sizeof(DCB);
    dcb.BaudRate = nBaud;          //ボーレート
    dcb.Parity = NOPARITY;        //パリティ：なし
    dcb.ByteSize = 8;             //ビット数：8
    dcb.StopBits = TWOSTOPBITS;   //ストップビット：1
    if (!SetCommState(hCom_1, &dcb))
    {
        printf("SetCommState() failed%cn");
        CloseHandle(hCom_1);
        return -1;
    }

    // 非同期送信を制御するための新しい OVERLAPPED 構造を作成します
    OVERLAPPED wrOverlapped;
    memset(&wrOverlapped, 0, sizeof(wrOverlapped));
    if (wrOverlapped.hEvent != NULL)
    {
        ResetEvent(wrOverlapped.hEvent);
    }
}
```

```
        wrOverlapped.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    }

    char DataBuffer[] = "This is some test data to write to the
virtual com port.¥r¥n";

    DWORD dwBytesToWrite = (DWORD)strlen(DataBuffer);
    DWORD dwBytesWritten = 0;
    // シリアルポートエラーとシリアルポート送信バッファをクリアします
    if (ClearCommError(hCom_1, &dwError, NULL))
    {
        PurgeComm(hCom_1, PURGE_TXABORT | PURGE_TXCLEAR);
    }

    // WriteFile を使用してデータを送信し、ループして送信が完了したかどうかを判断します
    if
    (!WriteFile(hCom_1, DataBuffer, dwBytesToWrite, &dwBytesWritten, &wrOverlapped))
    {
        if (GetLastError() == ERROR_IO_PENDING)
        {
            while
            (!GetOverlappedResult(hCom_1, &wrOverlapped, &dwBytesWritten, FALSE))
            {
                if (GetLastError() == ERROR_IO_INCOMPLETE)
                {
                    //Not finished, continue and get the result again
                    continue;
                }
            }
            else
            {

```

```
        printf("Send Error Occured¥n");
        ClearCommError(hCom_1, &dwError, NULL);
        break;
    }
}

printf("Data send finished, Bytes Written: %d¥n",
dwBytesWritten);
}
}

// 読み出しバッファをクリアします
PurgeComm(hCom_1, PURGE_RXCLEAR | PURGE_RXABORT);
//clear error
COMSTAT cs;
if(!ClearCommError(hCom_1, &dwError, &cs))
{
    printf("ClearCommError() failed¥n");
    CloseHandle(hCom_1);
    return -1;
}

SetCommMask(hCom_1, EV_RXCHAR); //データを受信するようにイベント
を設定します

//データを受信するための新しい補助スレッドを作成します
HANDLE hThread1 =
CreateThread(NULL, 0, ThreadRxMsg, NULL, 0, NULL);

while(1)
{
    // to do
}
```

```
CloseHandle(hCom_1);

return 0;
}

// シリアルポートデータを非同期で受信するための補助スレッド
DWORD WINAPI ThreadRxMsg(LPVOID lpParameter)
{
    while(1)
    {
        // 非同期受信を制御するための OVERLAPPED 構造
        OVERLAPPED osWait;

        memset(&osWait, 0, sizeof(OVERLAPPED));

        osWait.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

        DWORD dwEvtMask;

        if (WaitCommEvent(hCom_1, &dwEvtMask, &osWait))
        {
            //受信バッファが空でない場合、バッファ内のデータが処理されます
            if(dwEvtMask & EV_RXCHAR)
            {
                DWORD dwError;

                COMSTAT cs;

                if(!ClearCommError(hCom_1, &dwError, &cs))
                {
                    printf("ClearCommError() failed\n");

                    CloseHandle(hCom_1);

                    return -1;
                }
            }
        }
    }
}
```

```
char buf[101] = {0};

DWORD nLenOut = 0;

DWORD dwTrans = 0;

OVERLAPPED osRead;

memset(&osRead, 0, sizeof(OVERLAPPED));

osRead.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

BOOL bReadStatus =
ReadFile(hCom_1, buf, cs.cbInQue, &nLenOut, &osRead);

if(!bReadStatus)
{
    if(GetLastError() == ERROR_IO_PENDING)
    {
        printf("GetLastError() == ERROR_IO_PENDING¥n");

        // to do
    }
}

else
{
    printBuf(buf, nLenOut);
}

}

else
{
    if(GetLastError() == ERROR_IO_PENDING)
    {
        WaitForSingleObject(osWait.hEvent, INFINITE);

        if(dwEvtMask & EV_RXCHAR)
        {
            DWORD dwError;

            COMSTAT cs;
```

```
if(!ClearCommError(hCom_1,&dwError,&cs))
{
    printf("ClearCommError() failed¥n");
    CloseHandle(hCom_1);
    return -1;
}

char buf[101] = {0};
DWORD nLenOut = 0;
DWORD dwTrans = 0;
OVERLAPPED osRead;
memset(&osRead,0,sizeof(OVERLAPPED));
osRead.hEvent = CreateEvent(NULL,TRUE,FALSE,NULL);
BOOL bReadStatus =
ReadFile(hCom_1,buf,cs.cbInQue,&nLenOut,&osRead);
if(!bReadStatus)
{
    if(GetLastError() == ERROR_IO_PENDING)
    {
        printf("GetLastError() == ERROR_IO_PENDING¥n");
        // to do
    }
}
else
{
    printBuf(buf,nLenOut);
}
}
}
}
```

```
        return 1;
    }
}
```

// シリアルポート番号に従ってシリアルポート名を生成します。シリアルポート番号が 10 以上の場合、シリアルポート名は “¥¥¥¥.¥¥COMXX” の形式にする必要があります。

```
void getComName(char* ComNamePtr, unsigned int ComNum)
{
    if(ComNum > 9)
    {
        sprintf(ComNamePtr, "¥¥¥¥.¥¥COM%d", ComNum);
    }
    else
    {
        sprintf(ComNamePtr, "COM%d", ComNum);
    }

    return;
}
```

// 受信したシリアルデータを印刷します

```
void printBuf(char *buf, unsigned int bufSize)
{
    for(int i = 0; i < bufSize; i++)
    {
#ifdef PRINT_HEX
        printf("%02x ", buf[i]);
    }
    else
        printf("%c", buf[i]);
    }
}
```

```
    }  
  
    return;  
}
```

Windows 仮想シリアルポートの関数インターフェースの詳細については、Microsoft の[公式ドキュメント](#)を参照してください。

用語、略語

表 A-1 に、本マニュアルで使用される用語、略語、及びその意味を示します。

表 A-1 用語、略語

用語、略語	正式名称	意味
USB	Universal Serial Bus	ユニバーサル・シリアル・バス
UART	Universal Asynchronous Receiver/Transmitter	汎用非同期送受信回路

テクニカル・サポートとフィードバック

GOWIN セミコンダクターは、包括的な技術サポートをご提供しています。使用に関するご質問、ご意見については、直接弊社までお問い合わせください。

Web サイト : www.gowinsemi.com

E-mail : support@gowinsemi.com

